Synthesizing Verifiable Mathematical Problems

Anonymous ACL submission

Abstract

Mathematical data synthesis offers a potentially effective solution for enhancing the mathematical capabilities of large language models. However, existing methods either synthesize a large number of rationales based on existing questions, limiting the diversity of the questions, or rely on advanced proprietary models to directly generate new questions without verification, which cannot guarantee the correctness of the synthesized problems. This paper introduces a novel method, mathematical data synthesis through Algorithmic **A**bstraction, **I**mplementation, and **C**ontextualization (AIC), to synthesize new and verifiable mathematical problems. **AIC** abstracts mathematical problems into algorithms, implements these algorithms as code functions, and contextualizes them under different conditions to create new problems, which are then verified using code functions. Experimental results on multiple challenging mathematical benchmarks show that models fine-tuned on our synthesized data are superior to previous state-of-the-art models. Further experiments indicate that, when controlling for the same synthesizer, data synthesized using the AIC method is not only more accurate but also more effective at improving the model's mathematical abilities.

1 Introduction

Large language models (LLMs) have made significant advances, expanding beyond natural language processing into domains such as code generation and creative writing (Brown et al., 2020; Yuan et al., 2022; Chen et al., 2021; Guo et al., 2024). Their success is largely attributed to the vast quantities of high-quality training data available (Yuan et al., 2023; Gao et al., 2021). However, as the supply of untapped high-quality data diminishes, LLM research faces a growing challenge of data scarcity (Villalobos et al., 2022). Consequently, data synthesis, using generative models to create data similar to real data, offers a solution to this scarcity by supplementing real-world data (Liu et al., 2024; Bauer et al., 2024). For synthetic data to be effective, it should maintain quality comparable to real data (Xu et al., 2024; Van Breugel et al., 2023), particularly for mathematical data, which demands high logical consistency.

Table 1: Comprasion with other mathematical data synthesis methods."New Questions" represents whether the method generates logically distinct problems. "Verifiable" indicates whether the generated data are verifiable, for example, by using existing final answers for verification. "Synthesizer" refers to the model used for data synthesis.

Methods	New Questions	Verifiable	Synthesizer
MathCoder (Wang et al., 2023a)	✗	✓	GPT-4
MAmmoTH (Yue et al., 2023)	✗	✓	GPT-4
OpenMathInstruct (Toshniwal et al., 2024)	✗	✓	Mixtral
MetaMATH (Yu et al., 2023)	✓	✗	GPT-3.5
Abel (Chern et al., 2023)	✓	✗	GPT-4
MMIQC (Liu and Yao, 2024)	✓	✗	GPT-4
Orca-MATH (Mitra et al., 2024)	✓	✗	GPT-4
RFT (Yuan et al., 2023)	✗	✓	Llama2
Ours	✓	✓	Llama3 + DSMath

Current approaches to enhancing model capabilities via data synthesis fall into two distinct categories: (1) methods that generate novel rationales for existing problems without creating new problems, and (2) methods that produce new problems whose correctness cannot be verified. The first approach leverages LLMs' problem-solving abilities to generate rationales for existing mathematical problems with known answers, employing rejection sampling (Yuan et al., 2023) to filter rationales based on the correctness of the final answer (Wang et al., 2023a; Gou et al., 2023; Toshniwal et al., 2024; Tong et al., 2024). While effective, these methods overlook problem diversity, potentially limiting the model's generalization ability. The second approach utilizes advanced LLMs, such as GPT-4 (Achiam et al., 2023), to generate entirely new questions and corresponding rationales, significantly improving data diversity. (Chern et al., 2023; Luo et al., 2023; Liu and Yao, 2024; Tang et al., 2024; Huang et al., 2024; Li et al., 2024).

Nevertheless, even the most advanced models are prone to errors, and direct synthesis without verification raises concerns about data correctness (Yu et al., 2023; Mitra et al., 2024). Therefore, a method that generates new problems while ensuring their correctness is essential for producing diverse and accurate synthetic mathematical data.

In this paper, we introduce Mathematical Data Synthesis via Algorithmic Abstraction, Implementation, and Contextualization (AIC). The core idea is that most mathematical problems can be addressed through algorithms, such as determining the greatest common divisor (GCD) of two integers, which can be accomplished using the well-established Euclidean algorithm. By abstracting such algorithms from existing mathematical problems and contextualizing them under different conditions, we can generate novel mathematical questions and corresponding rationales. Furthermore, these abstract mathematical algorithms can be implemented in Python to verify the correctness of the synthesized data. As illustrated in Figure 1, the AIC process comprises two stages: **Stage 1: Algorithm Abstraction and Implementation** First, we employ LLMs as synthesizers to abstract existing mathematical problems, which serve as seed data. Each entry, consisting of a question, rationale, and final solution, is transformed into a natural language algorithm. Subsequently, we prompt the synthesizer to implement the algorithm as a Python code function and verify its correctness using a verification mechanism. **Stage 2: Algorithm Contextualization** The synthesizer contextualizes the natural language algorithm under different conditions to generate a new mathematical problem. The conditions of this newly generated problem are then input into the corresponding code function. By comparing the final answer generated by the synthesizer with the result of the code execution, we verify the correctness of the synthesized problem.

The experiments demonstrate the effectiveness of AIC in mathematical data synthesis. Using MATH (Hendrycks et al., 2021) and a subset of MAmmoTH2 (Yue et al., 2024) as seed data, we employ AIC to generate numerous new mathematical problems, resulting in the AIC dataset comprising 1.3M samples, including 240k unique questions. Fine-tuning various models of different scales on AIC outperformed state-of-the-art methods by an average of 7.1% across multiple challenging benchmarks including MATH (Hendrycks et al., 2021), Gaokao-Math (Zhang et al., 2023), MathOdyssey (Fang et al., 2024), OlympaidBench-Math (He et al., 2024), and TheoremQA (Chen et al., 2023). Notably, DSMath-7B-AIC, a fine-tuned version of DeepSeekMath-7B-Base on the AIC dataset, achieved 51.0% accuracy on MATH, surpassing previous state-of-the-art model DSMath-7B-MMIQC (Liu and Yao, 2024) by 8.5%. Furthermore, by controlling the synthesizer and data scale, we demonstrated that our pipeline outperforms alternative methods. Rigorous ablation studies confirmed the significance of our proposed verification mechanism in ensuring data correctness. Our approach initiates the first step in exploring verifiable methods of data synthesis.

2 Related Work

Researchers have employed various approaches to enhance the mathematical reasoning capabilities of LLMs. One strand of research focuses on model training, with the primary challenge being acquiring suitable training data. Some studies have addressed this by classifying, cleaning, and filtering internet corpora to obtain high-quality mathematical content for continual pretraining or post-training (Paster et al., 2023; Wang et al., 2023b; Shao et al., 2024; Zhou et al., 2024). Others have explored the synthesis of new instruction-tuning data based on existing models and datasets (Yu et al., 2023; Luo et al., 2023; Fan et al., 2024). A second research direction concentrates on optimizing LLM output during inference, either by improving reasoning accuracy through search algorithms or by refining the reasoning process using feedback mechanisms (Wang et al., 2022; Yao et al., 2024; Lightman et al., 2023; Madaan et al., 2023). This paper primarily investigates methods for synthesizing mathematical reasoning data for instruction fine-tuning to enhance the mathematical capabilities of LLMs.

Rationale Generation Methods such as RFT (Yuan et al., 2023), STaR (Zelikman et al., 2022), and DART-Math (Tong et al., 2024) leverage LLMs' problem-solving abilities by generating diverse chain-of-thought rationales for existing question-answer pairs and verifying the reasoning process by checking the final answer, known as rejection sampling. ToRA (Gou et al., 2023), MathCoder (Wang et al., 2023a), and OpenMathInstruct-1 (Toshniwal et al., 2024) distill code-interpreter format rationales from LLMs, using code to mitigate hallucinations in LLM

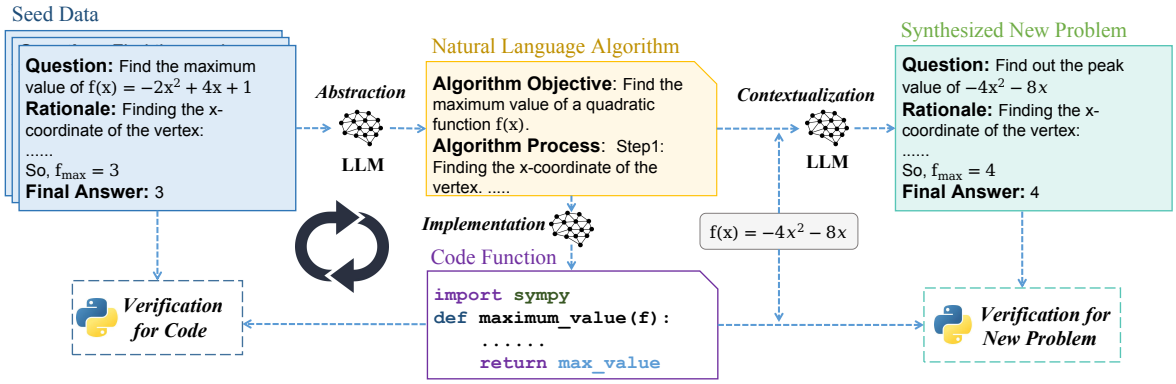

Figure 1: An overview of AIC: (1) LLMs as synthesizers to abstract existing data problems (Seed Data) into natural language algorithms. (2) LLM then implements these natural language algorithms as Python code, subsequently verifying the code's correctness through verification. (3) Finally, the LLM contextualizes the abstract algorithms to generate new problems, utilizing a verification mechanism to ensure the correctness of the newly created data.

calculations and improve reasoning accuracy. However, these approaches do not generate new questions, potentially limiting question diversity and model generalization.

Question Synthesis Recent studies have extensively explored using LLMs to generate novel mathematical questions and their corresponding rationales. Notable studies in this field include WizardMath (Luo et al., 2023), MMIQC (Liu and Yao, 2024), MuggleMath (Li et al., 2023), MathScale (Tang et al., 2024), KPMath (Huang et al., 2024), Xwin-Math (Li et al., 2024), and Orca-Math (Mitra et al., 2024). Since this demands a high level of mathematical capability in the synthesized model, they all use closed-source advanced models (e.g., GPT-3.5, GPT-4) for synthesis. However, methods that directly require LLMs to generate new questions lack effective verification mechanisms for the correctness of rationales and final answers, making errors in the synthesized data inevitable. MetaMath (Yu et al., 2023) uses bootstrapping by backward reasoning to synthesize verifiable new problems, but the assumptions of this method are often invalid for difficult problems, leading to logical errors in the synthesis of complex problems.

3 Method

In this paper, we propose a data synthesis method for generating new mathematical problems with verified solutions. Our method comprises two stages. In the first stage, we employ a LLM as a synthesizer to abstract core algorithms from existing mathematical problems. We then prompt the synthesizer to implement these algorithms as Python code functions, incorporating existing libraries (e.g., SymPy, SciPy, NumPy), and verify the code's correctness. In the second stage, the synthesizer contextualizes the algorithms into new mathematical problems, using the code functions to verify the correctness of the synthesized problems.

Let $\mathcal{D}_{\text{seed}} = (q_i, r_i, a_i)_{i=1}^{N}$ denote a typical mathematical training dataset, serving as seed data for our synthesis process. Each entry in $\mathcal{D}_{\text{seed}}$ consists of three components: the question q_i, the rationale r_i, and the answer a_i. In addition to the seed data, the synthesis process requires LLMs $\mathcal{M}$ (e.g., Llama3-70B-Instruct, Mixtral-8x7B-Instruct) as synthesizers and a code interpreter $\mathcal{C}$. The synthesizer $\mathcal{M}$ generates text, including natural language algorithms, code, and new problems, while the code interpreter $\mathcal{C}$ executes code to verify the correctness of both the code functions and the synthesized problems.

3.1 Stage1: Algorithm Abstraction and Implementation

Algorithm Abstraction Algorithm abstraction involves identifying the core logic of a problem and transforming specific questions and their solutions into generalized, reusable algorithms. These algorithms can then be used to generate entirely new mathematical problems. For each data point $d_i = (q_i, r_i, a_i)$, the synthesizer $\mathcal{M}$ analyzes the question q_i and rationale r_i to understand the problem's core goals, identify key operations and reasoning steps, determine sequential relationships between steps, and identify mathematical objects (e.g., integers, series, expressions) in q_i that are independent of the core reasoning steps. These objects are then parameterized as placeholder vari-

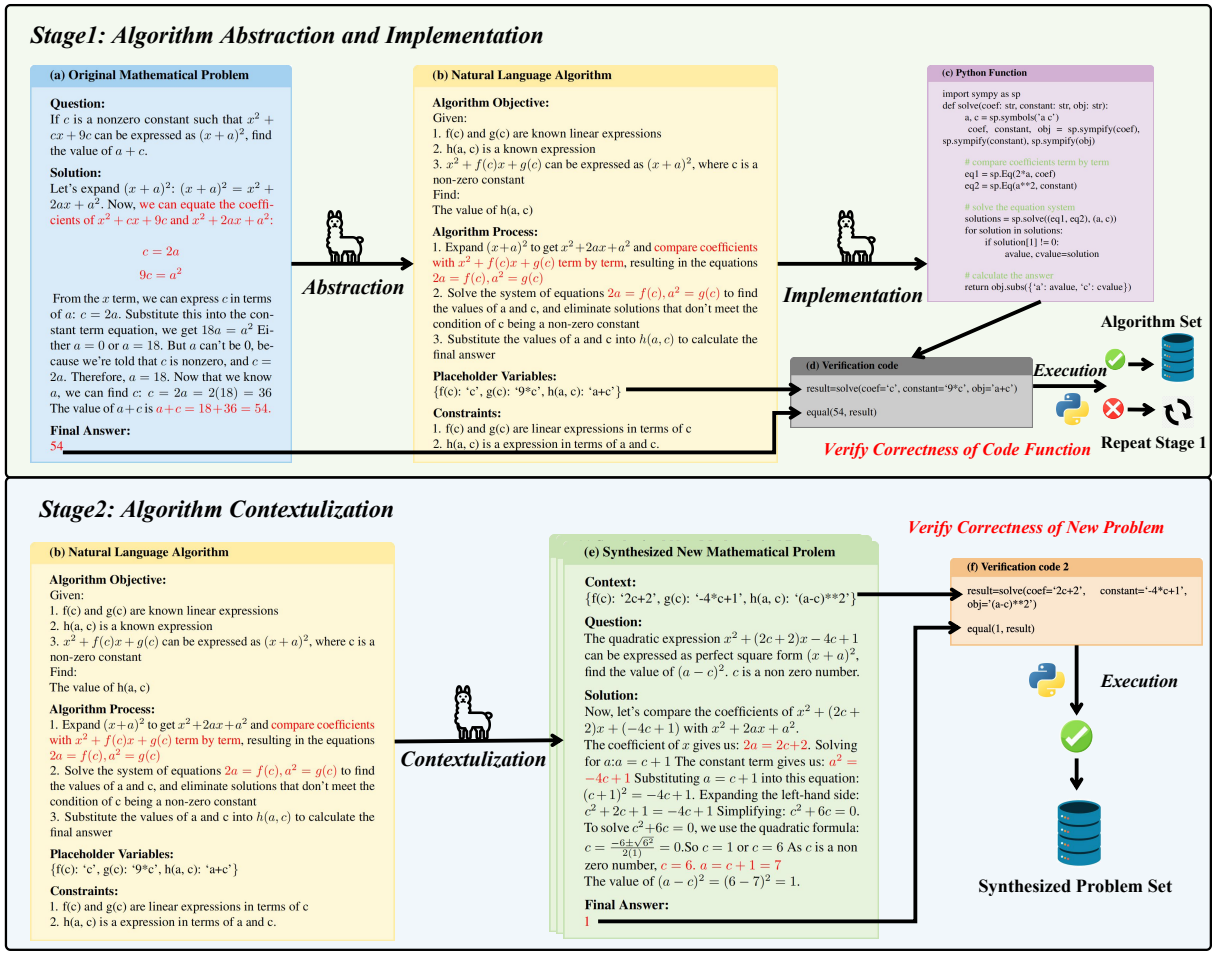

Figure 2: The pipeline of Algorithm Abstraction, Implementation, and Contextualization. We emphasize the central components for illustrating mathematical problems and algorithms.

ables. Through this process, a question q_i and its rationale r_i are transformed into an **algorithm objective** o_i and an **algorithm process** p_i. For example, in the problem of finding the greatest common divisor of 14 and 42, the abstracted algorithm objective is to find the GCD of a and b, with the algorithm process being the Euclidean algorithm. The synthesizer $\mathcal{M}$ also generates additional information about the algorithm, including **placeholder constraints** c_i, which specify the mathematical object types of the placeholder variables and their value ranges, and **placeholder values** v_i, which indicate the values of the placeholder variables in the original problem. Using appropriate instructions and examples, we prompt the synthesizer $\mathcal{M}$ to abstract a mathematical problem $d_i = (q_i, r_i, a_i)$ into a natural language algorithm $\Psi_i = (o_i, p_i, c_i, v_i)$.

Algorithm Implementation Algorithm implementation aims to convert abstract natural language mathematical algorithms into concrete, executable Python code, which will be used to verify the correctness of the synthesized data. For each algorithm $\Psi_i = (o_i, p_i, v_i, c_i)$, the synthesizer $\mathcal{M}$ is tasked with creating a Python function f_i using existing mathematical libraries (e.g., NumPy, SymPy, SciPy). The function parameters are the **placeholder variables**, and the return value is the final result of the algorithm. To ensure the correctness of the algorithm and function, we propose a verification mechanism that uses the original problem $d_i = (q_i, r_i, a_i)$ as a test case. By inputting placeholder values into the function f_i, obtaining the function's return value using code interpreter $\mathcal{C}$, and comparing it with the answer a_i from the original problem, we can filter out incorrect functions. Algorithms that pass verification are added to the algorithm set for subsequent synthesis of new problems. If verification fails, we leverage the automated algorithm generation and verification process to regenerate the algorithm for the problem, repeating until the algorithm passes verification or reaches the maximum number of iterations I.

3.2 Stage2: Contextualization

Contextualization aims to transform abstract natural language algorithms into specific mathematical problems and verify the correctness of synthesized data through code execution. For a given algorithm $\Psi_i = (o_i, p_i, v_i, c_i)$ and its corresponding code function f_i, we first prompt the synthesizer $\mathcal{M}$, using appropriate instructions and demonstrations, to generate K possible **new** values for placeholder variables $pv_i^j, j = 1, \ldots, K$ based on the algorithm. This process assigns specific mathematical objects that comply with the algorithm's constraints to the placeholder variables, effectively creating a context. With the algorithm and placeholder variables defined, the synthesizer $\mathcal{M}$ then synthesizes specific mathematical questions q_i^j, corresponding rationales r_i^j, and final answers a_i^j for the algorithm in each context pv_i^j. Unlike traditional synthesis algorithms (Liu and Yao, 2024; Tang et al., 2024) that lack verification methods or rely on unreliable verification, our approach employs a robust verification process. We input the placeholder variable values pv_i^j into the code function f_i and execute it using the code interpreter $\mathcal{C}$. By comparing the execution result g_i^j with the final answer a_i^j provided by $\mathcal{M}$ through algorithm contextualization, we can filter out incorrect synthesized data. Through this process of generating numerous contexts, an algorithm can be contextualized into many mathematical problems, ensuring both diversity and correctness in the resulting dataset.

Furthermore, the synthesized a large number of entirely new mathematical question-answer pairs serve as a foundation for further dataset expansion through rejection sampling (Yuan et al., 2023), enabling the generation of diverse rationales for individual questions.

4 Experiments

4.1 Experiments Setup

Data We use the MATH dataset (Hendrycks et al., 2021), comprising 7,500 training samples, and a subset of MAmmoTH2 (Yue et al., 2024), containing approximately 30,000 mathematics-related data points, as seed data for our data synthesis process. For the synthesis model, we primarily employ the Llama3-70B-Instruct algorithm for abstraction, implementation, and contextualization. Subsequently, we use DeepSeekMath-7B-RL (Shao et al., 2024) to perform rejection sampling to synthesize diverse rationales. In some experiments, we also substitute Llama3-70B-Instruct with Mixtral-8×7B-Instruct (Jiang et al., 2024), to better simulate the scenario of using weak models to synthesize difficult problems.

Table 2: Statistics of the AIC Dataset.

Data	MATH	MAmmoTH2	Overall
# Seed Data	7,500	33,343	40,843
# Unique Question	103,879	134,100	237,979
# Synthesized Data	569,178	729,900	1,299,078

Using the seed data, we initially synthesize about 240k unique questions. The DeepSeekMath-7B-RL is used to generate 8 rationales for each question, filtering for those with correct answers. This process resulted in the AIC dataset, containing 1.3 million data points. The statistics of the AIC dataset are presented in Table 2.

Training We performed standard supervised fine-tuning on multiple models using the AIC dataset, including Llama3-8B-Base (Dubey et al., 2024), Mistral-7B-Base (Jiang et al., 2023), DeepSeekMath-7B-Base (Shao et al., 2024), and Llama3-70B-Base. Detailed information about the training, including training templates and hyperparameter settings, is provided in appenidx B.1.

Baseline In §4.2, we compare various mathematical datasets, including MetaMATH, WizardMATH, Abel, MMIQC, and MathScale. We reproduced the results of the open-source datasets MetaMATH and MMIQC. For the other datasets, we conducted evaluations using the officially released models. In §5.1, we compare different data synthesis methods under controlled conditions, including (1) NumReplace (Li et al., 2023), (2) MMIQC, (3) Xwin-Math, and (4) RFT. Detailed information about these methods is presented in appendix B.3.

Evaluation We evaluate model performance on multiple mathematical datasets through (1) in-domain evaluation using the MATH (Hendrycks et al., 2021) test set; and (2) out-of-distribution evaluation using several high-difficulty benchmarks, including GaokaoBench-Math (Zhang et al., 2023), MathOdyssey (Fang et al., 2024), OlympiadBench-Math (He et al., 2024), and TheoremQA (Chen et al., 2023). Detailed information about these benchmarks is provided in Appendix B.4.

Table 3: Comparison of different models testing accuracy on mathematical benchmarks. The prefix and suffix of the model name respectively represent the base model and the dataset. DSMath stands for DeepSeekMath-7B-Base, while Seed indicates Seed Data. The increase in accuracy compared to the previous best fine-tuned models is highlighted in green. Result on more benchmarks are shown in appendix F.

Model	Synthesizer	MATH	GaoKao	Odyssey	Olympiad	TheoremQA	Avg
Proprietary Models							
GPT4o	-	76.6	-	-	-	-	-
Claude 3.5 sonnet	-	71.1	-	-	-	-	-
Open-source models based on Llama3-70B-Base							
Llama3-70B-Base	-	38.3	22.8	26.6	13.9	22.2	24.8
Llama3-70B-Seed	-	38.6	23.2	11.4	10.1	16.5	20.0
Llama3-70B-Abel	GPT4	41.7	27.0	10.6	11.6	14.4	21.1
Llama3-70B-MetaMATH	GPT3.5	41.3	26.4	6.5	10.7	14.0	19.8
Llama3-70B-MMIQC	GPT4	44.5	20.9	11.1	9.3	10.2	19.2
Llama3-70B-AIC(Ours)	Llama3 + DSMath	**52.0**$_{+7.5}$	**31.9**$_{+4.9}$	**21.6**$_{+10.2}$	**16.3**$_{+4.7}$	**21.1**$_{+4.6}$	**28.6**$_{+7.5}$
Open-source models based on DeepSeekMath-7B-Base							
DSMath-7B-Base	-	30.1	15.4	18.6	8.4	13.5	17.2
DSMath-7B-Seed	-	33.7	24.2	7.8	8.0	13.1	17.4
DSMath-7B-Abel	GPT4	36.6	27.6	6.2	8.7	13.0	18.4
DSMath-7B-MetaMATH	GPT3.5	41.8	27.9	8.3	10.1	14.4	20.5
DSMath-7B-MMIQC	GPT4	42.7	19.1	17.8	9.3	10.2	19.6
DSMath-7B-AIC(Ours)	Llama3 + DSMath	**51.0**$_{+8.3}$	**39.0**$_{+11.1}$	**18.6**$_{+0.8}$	**18.1**$_{+8.0}$	**23.1**$_{+8.7}$	**30.0**$_{+9.5}$
Open-source models based on Mistral-7B-Base							
Mistral-7B-Base	-	11.0	4.4	7.5	2.1	6.4	6.3
Mistral-7B-Seed	-	18.0	9.5	4.9	2.8	10.0	9.0
Mistral-7B-Abel	GPT4	24.4	14.4	6.5	5.3	10.2	13.0
Mistral-7B-WizardMATH	GPT4	32.3	17.1	5.4	7.4	12.4	14.9
Mistral-7B-MetaMATH	GPT3.5	27.7	14.9	5.9	6.5	6.0	12.2
Mistral-7B-MMIQC	GPT4	36.5	17.9	7.2	6.8	9.2	15.4
Mistral-7B-MathScale	GPT4	35.2	21.4	3.9	6.7	7.4	14.9
Mistral-7B-AIC(Ours)	Llama3 + DSMath	**41.4**$_{+5.4}$	**22.7**$_{+1.3}$	**11.6**$_{+4.4}$	**13.5**$_{+6.1}$	**16.1**$_{+3.7}$	**21.1**$_{+5.7}$
Open-source models based on Llama3-8B-Base							
Llama3-8B-Base	-	19.4	8.1	16.3	4.6	10.5	11.8
Llama3-8B-Seed	-	19.9	9.7	3.6	2.8	9.5	9.1
Llama3-8B-Abel	GPT4	27.5	15.4	7.0	5.0	10.0	13.0
Llama3-8B-MetaMATH	GPT3.5	31.5	14.7	6.4	6.8	10.2	13.9
Llama3-8B-MMIQC	GPT4	37.5	15.3	11.3	6.9	9.7	16.1
Llama3-8B-AIC(Ours)	Llama3 + DSMath	**43.1**$_{+5.6}$	**27.2**$_{+11.8}$	**12.9**$_{+1.6}$	**10.8**$_{+3.9}$	**14.5**$_{+4.3}$	**21.7**$_{+5.6}$

4.2 Verify Effectiveness of the AIC Dataset

The results presented in Table 3 demonstrate that models fine-tuned on the AIC dataset consistently outperform those fine-tuned on various other mathematical datasets across multiple base models. Notably, DSMath-7B-AIC achieves an average improvement of 9.5% compared to other models fine-tuned on DeepSeekMath-7B-Base across various benchmarks. Similarly, Llama3-70B-AIC, Mistral-7B-AIC, and Llama3-8B-AIC show average improvements of 7.5%, 5.7%, and 5.6%, respectively, compared to their corresponding fine-tuned counterparts. For the in-domain MATH benchmark, models trained on the AIC dataset demonstrate consistent gains over other models. DSMath-7B-AIC outperforms the state-of-the-art model DSMath-7B-Metamath by 8.3%. Additionally, Llama3-70B-AIC, Mistral-7B-AIC, and Llama3-8B-AIC show improvements of 7.5%, 5.4%, and 5.6%, respectively. The AIC dataset also effectively enhances model performance on out-of-distribution benchmarks. Llama3-8B-AIC outperforms Llama3-8B-Abel by 11.8% on GaoKaoBench-Math, while DSMath-7B-AIC demonstrates an 8.0% improvement over DSMath-7B-MMIQC on OlympiadBench-Math.

4.3 Impact of Seed Data

In this section, we split the synthetic data according to seed data, with the portion synthesized based on MATH referred to as AIC-MATH, while the data

Table 4: The effectiveness of different portions is evaluated. MAmmoTH2 represents the subset of the MAmmoTH2 dataset used as seed data, comprising approximately 30,000 data points.

Data	MATH	GaoKao	Odyssey	Olympiad	TheoremQA
Mistral-7B-Base					
MATH	15.6	9.8	4.4	3.3	9.3
MAmmoTH2	17.7	11.2	4.9	4.4	10.3
AIC-MATH	**38.8**	**22.3**	10.3	**9.5**	13.3
AIC-MAmmoTH2	29.9	20.1	**16.3**	7.0	**16.0**
Llama3-8B-Base					
MATH	17.9	9.3	3.1	1.9	6.1
MAmmoTH2	18.8	11.4	5.2	3.4	9.4
AIC-MATH	**40.7**	**21.5**	9.8	**11.1**	13.8
AIC-MAmmoTH2	30.4	20.3	**14.2**	7.0	**12.4**

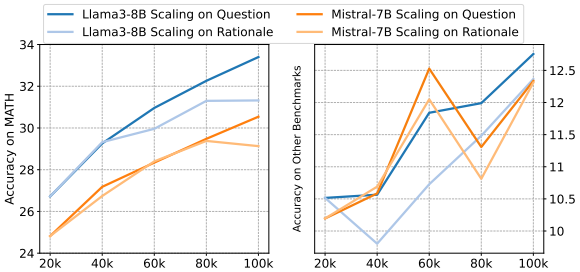

Figure 3: Scaling effects on questions and rationales. The accuracy of other benchmarks is the average accuracy of four OOD benchmarks.

synthesized based on MAmmoTH2 is called AIC-MAmmoTH2. We fine-tuned both portions of data on Mistral-7B-Base and Llama3-8B-Base, and the results displayed in Table 4 indicate that both data synthesized based on MATH and MAmmoTH2 lead to performance improvements across all benchmarks. It is also clear that when using MATH as seed data, the synthesized data shows greater improvement on the MATH test set, while using MAmmoTH2 as seed data yields significant performance gains on MathOdyssey and TheoremQA.

4.4 Scaling Effects on Question and Rationale

We investigated the different impacts on model performance between two data synthesis approaches: generating brand new questions and generating diverse rationales for existing questions. To facilitate this analysis, we derived two distinct datasets from the AIC dataset. The first dataset comprised 20k unique questions, each associated with multiple rationales, and was utilized to examine the scaling effects of rationales. The questions in the second dataset are distinct from each other, each paired with a single rationale, and were employed to study the scaling effects of questions. We fine-tuned the model using these two types of datasets

Table 5: Ablation study of verification mechanism. Verification 1 represents the verification for code in stage 1 and verification 2 represents the verification for new problem in stage 2.

Verfication 1	Verfication 2	MATH	
		Llama3-8B-Base	Mistral-7B-Base
✓	✓	$21.2_{+1.3}$	$18.8_{+1.5}$
✓	✗	$20.1_{+0.2}$	$17.8_{+0.5}$
✗	✓	$20.4_{+0.5}$	$16.7_{-0.6}$
✗	✗	19.9	17.3

across various scales (20k, 40k, 60k, 80k, and 100k samples). The results, presented in Figure 3, demonstrate that increasing the number of unique questions consistently yielded greater performance improvements than adding more rationales to existing questions. This trend was observed across both the in-domain MATH benchmark and out-of-domain (OOD) benchmarks and persisted regardless of whether the fine-tuning was performed on Mistral-7B-Base or Llama3-8B-Base. These findings underscore the significance of synthesizing novel problems rather than merely augmenting existing questions with additional rationales.

4.5 Effectiveness of Verification Mechanism

We investigated the impact of verification mechanisms on the data synthesis process, including verification for code in stage 1 and verification for new problem in stage 2. Using the MATH dataset as seed data and Mixtral-8×7B-Instruct as synthesizer, we constructed four datasets based on the AIC pipeline (not applying rejection sampling), varying the application or not of verification for code and new problems. These datasets were filtered to the same scale of about 34k samples. Results in Table 5 indicate that implementing both verification mechanisms significantly enhances data quality compared to no verification, yielding over a 1% performance improvement in the fine-tuned model, even with a limited scale dataset. Notably, applying only one verification mechanism produced substantially less improvement than employing both. The efficacy of the verification mechanism stems from ensuring the correctness of code-generated final answers and filtering rationales based on these answers, thereby enhancing the logical and computational accuracy of the data. These findings underscore the critical role of verification in synthesized data quality assurance.

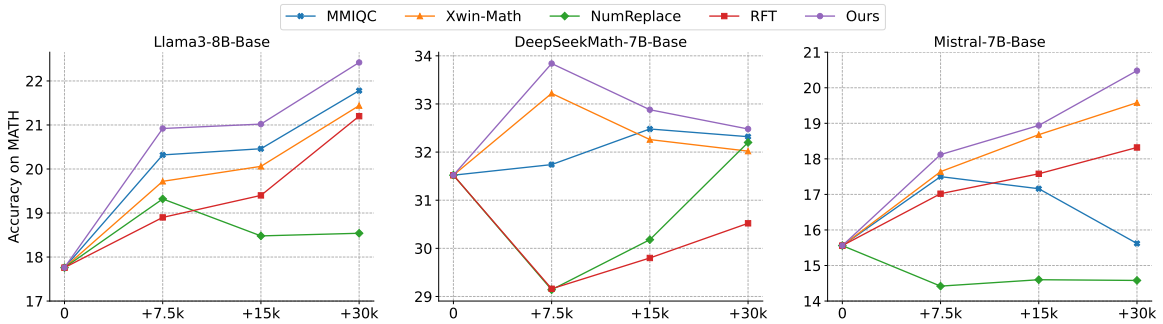

Figure 4: The results of a fair comparison of other data synthesis methods. The horizontal axis represents the scale of the synthesized data, and we used the synthesized data along with the original MATH training set for fine-tuning.

5 Analysis

5.1 Controllable Comparison with Other Methods

Given the variation in both the synthesizer and the scale of data synthesized across the comparison objects in § 4.2, these differences do not accurately represent the strengths and weaknesses of the synthesizer. So we standardize both the synthesizer and the scale of data synthesized in this section, allowing for a more comprehensive evaluation of the methods. We chose MATH as the seed data, Mixtral-8×7B-Instruct as the synthesizer, and conducted all evaluations on MATH. The methods compared include NumReplace, MMIQC, Xwin-MATH, and RFT as introduced in Appendix B.3. We trained models on various data scales to thoroughly assess the effectiveness of the synthesis methods. The results in Figure 4 show that our method outperforms the baselines at any scale on various base models. We believe this is because, when generating more difficult problems, methods without a verification mechanism often lead to errors, thereby reducing data quality.

5.2 Pass Ratio of Verification Mechanism

We present the pass ratio of two verification mechanisms: verification for code and verification for new problems. For verification for code, by attempting to abstract and implement a problem more times, we can improve the success rate of converting a problem into an algorithm that passes verification, which is the pass ratio of verification for code. As for verification for new problems, an algorithm can be contextualized multiple times under different conditions, resulting in different problems. We define its pass ratio as the proportion of algorithms where at least one of the contextualizations leads to a problem that passes verification. The result in

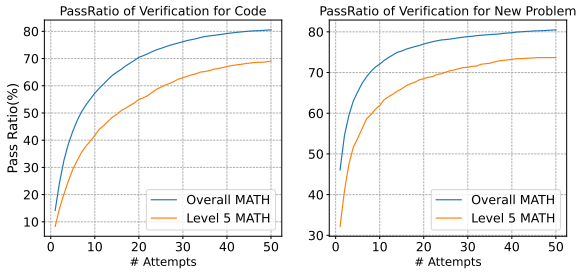

Figure 5: Pass Ratio of Verification Mechanism on overall MATH and Level5 MATH.

Figure 5 illustrates that for both verification mechanisms, the pass rate across the entire MATH dataset hovers around 80%, and it has not yet reached a growth plateau. On the other hand, under both verification mechanisms, the pass rate for Level 5 is lower, but it still achieves around 70%.

5.3 Failure Cases

We also explored, through several examples, the circumstances under which the code and new problems generated by the synthesizer fail to pass the two verification mechanisms. The specific examples placed in appendix E show that inaccurate algorithm abstraction affects subsequent code implementation and contextualization, further leading to the failure of verification.

6 Conclusion

This paper introduces a mathematical data synthesis methodology that generates novel problems and verifies their correctness through algorithm abstraction, implementation, and contextualization of existing mathematical problems. Empirical evaluations across multiple benchmarks demonstrate the efficacy of our approach. Ablation studies further emphasize the importance of synthesizing entirely new problems and ensuring their correctness.

7 Limitation

This study focused on synthesizing data to enhance models' mathematical capabilities. However, we envision that AIC could also improve large models' performance in other reasoning tasks, particularly those that can be abstracted into algorithms, and implemented through code, such as the 24-point game. Additionally, while our current algorithm can only validate answers to newly generated questions, validating the intermediate steps of the rationale could further improve data quality.

8 Ethical Considerations

The accuracy and reliability of the synthesized answers are paramount. If LLMs produce incorrect or misleading solutions, it could undermine educational integrity and lead to misconceptions in mathematical understanding. Therefore, it is essential to implement rigorous validation mechanisms to ensure the correctness of the generated answers.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *ArXiv preprint*, abs/2303.08774.

André Bauer, Simon Trapp, Michael Stenger, Robert Leppich, Samuel Kounev, Mark Leznik, Kyle Chard, and Ian Foster. 2024. Comprehensive exploration of synthetic data generation: A survey. *ArXiv preprint*, abs/2401.02524.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *ArXiv preprint*, abs/2107.03374.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. 2023. Theoremqa: A theorem-driven question answering dataset. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7889–7901.

Ethan Chern, Haoyang Zou, Xuefeng Li, Jiewen Hu, Kehua Feng, Junlong Li, and Pengfei Liu. 2023. Generative ai for math: Abel. https://github.com/GAIR-NLP/abel.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *ArXiv preprint*, abs/2407.21783.

Run-Ze Fan, Xuefeng Li, Haoyang Zou, Junlong Li, Shwai He, Ethan Chern, Jiewen Hu, and Pengfei Liu. 2024. Reformatted alignment. *arXiv preprint arXiv:2402.12219*.

Meng Fang, Xiangpeng Wan, Fei Lu, Fei Xing, and Kai Zou. 2024. Mathodyssey: Benchmarking mathematical problem-solving skills in large language models using odyssey math data. *arXiv preprint arXiv:2406.18321*.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2021. The pile: An 800gb dataset of diverse text for language modeling. *ArXiv preprint*, abs/2101.00027.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. 2023. Tora: A tool-integrated reasoning agent for mathematical problem solving. *ArXiv preprint*, abs/2309.17452.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *Preprint*, arXiv:2402.14008.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *ArXiv preprint*, abs/2103.03874.

Yiming Huang, Xiao Liu, Yeyun Gong, Zhibin Gou, Yelong Shen, Nan Duan, and Weizhu Chen. 2024. Key-point-driven data synthesis with its enhancement on mathematical reasoning. *ArXiv preprint*, abs/2403.02333.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *ArXiv preprint*, abs/2401.04088.

Chen Li, Weiqi Wang, Jingcheng Hu, Yixuan Wei, Nanning Zheng, Han Hu, Zheng Zhang, and Houwen Peng. 2024. Common 7b language models already possess strong math capabilities. *ArXiv preprint*, abs/2403.04706.

Chengpeng Li, Zheng Yuan, Guanting Dong, Keming Lu, Jiancan Wu, Chuanqi Tan, Xiang Wang, and Chang Zhou. 2023. Query and response augmentation cannot help out-of-domain math reasoning generalization. *ArXiv preprint*, abs/2310.05506.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *ArXiv preprint*, abs/2305.20050.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*.

Haoxiong Liu and Andrew Chi-Chih Yao. 2024. Augmenting math word problems via iterative question composing. *ArXiv preprint*, abs/2401.09003.

Ruibo Liu, Jerry Wei, Fangyu Liu, Chenglei Si, Yanzhe Zhang, Jinmeng Rao, Steven Zheng, Daiyi Peng, Diyi Yang, Denny Zhou, et al. 2024. Best practices and lessons learned on synthetic data for language models. *ArXiv preprint*, abs/2404.07503.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *ArXiv preprint*, abs/2308.09583.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *Preprint*, arXiv:2303.17651.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing english math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984.

Arindam Mitra, Hamed Khanpour, Corby Rosset, and Ahmed Awadallah. 2024. Orca-math: Unlocking the potential of slms in grade school math. *ArXiv preprint*, abs/2402.14830.

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. 2023. Openwebmath: An open dataset of high-quality mathematical web text. *ArXiv preprint*, abs/2310.06786.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Yu Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *ArXiv preprint*, abs/2402.03300.

Zhengyang Tang, Xingxing Zhang, Benyou Wan, and Furu Wei. 2024. Mathscale: Scaling instruction tuning for mathematical reasoning. *ArXiv preprint*, abs/2403.02884.

Yuxuan Tong, Xiwen Zhang, Rui Wang, Ruidong Wu, and Junxian He. 2024. Dart-math: Difficulty-aware rejection tuning for mathematical problem-solving. *ArXiv preprint*, abs/2407.13690.

Shubham Toshniwal, Ivan Moshkov, Sean Narenthiran, Daria Gitman, Fei Jia, and Igor Gitman. 2024. Openmathinstruct-1: A 1.8 million math instruction tuning dataset. *ArXiv preprint*, abs/2402.10176.

Boris Van Breugel, Zhaozhi Qian, and Mihaela Van Der Schaar. 2023. Synthetic data, real errors: how (not) to publish and use synthetic data. In *International Conference on Machine Learning*, pages 34793–34808. PMLR.

Pablo Villalobos, Jaime Sevilla, Lennart Heim, Tamay Besiroglu, Marius Hobbhahn, and Anson Ho. 2022. Will we run out of data? an analysis of the limits of scaling datasets in machine learning. *ArXiv preprint*, abs/2211.04325.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2023a. Mathcoder: Seamless code integration in llms for enhanced mathematical reasoning. *ArXiv preprint*, abs/2310.03731.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *ArXiv preprint*, abs/2203.11171.

Zengzhi Wang, Rui Xia, and Pengfei Liu. 2023b. Generative ai for math: Part i–mathpile: A billion-token-scale pretraining corpus for math. *ArXiv preprint*, abs/2312.17120.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. 2024. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing. *ArXiv preprint*, abs/2406.08464.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. Metamath: Bootstrap your own mathematical questions for large language models. *ArXiv preprint*, abs/2309.12284.

Ann Yuan, Andy Coenen, Emily Reif, and Daphne Ippolito. 2022. Wordcraft: story writing with large language models. In *Proceedings of the 27th International Conference on Intelligent User Interfaces*, pages 841–852.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *ArXiv preprint*, abs/2308.01825.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2023. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653*.

Xiang Yue, Tuney Zheng, Ge Zhang, and Wenhu Chen. 2024. Mammoth2: Scaling instructions from the web. *ArXiv preprint*, abs/2405.03548.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488.

Xiaotian Zhang, Chunyang Li, Yi Zong, Zhengyu Ying, Liang He, and Xipeng Qiu. 2023. Evaluating the performance of large language models on gaokao benchmark.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2023. Agieval: A human-centric benchmark for evaluating foundation models. *arXiv preprint arXiv:2304.06364*.

Fan Zhou, Zengzhi Wang, Qian Liu, Junlong Li, and Pengfei Liu. 2024. Programming every example: Lifting pre-training data quality like experts at scale. *arXiv preprint arXiv:2409.17115*.

A Data

A.1 Seed Data

We utilized the MATH training set and a carefully selected subset of MAmmoTH2 as the seed data for our synthesis process. The MATH dataset comprises competition-level mathematics problems, encompassing a broad spectrum of topics including algebra, geometry, probability, and number theory. In contrast, MAmmoTH2 is an instruction-tuned dataset derived from retrieved, cleaned, and rewritten mathematical content sourced from the internet, containing a vast array of mathematical problems.

The MATH training set, consisting of 7,500 problems, was incorporated in its entirety as seed data. From the expansive MAmmoTH2 dataset, which contains 10 million entries (of which 2 million are open-sourced), we extracted 30,000 high-quality examples related to mathematics. To ensure the utmost quality and accuracy of our seed data, we employed LLMs to filter these examples, prioritizing those with explicit answers and superior quality.

B Experiments Setup

B.1 Training

For standard supervised fine-tuning of both the Base and Instruct models, we employed the template illustrated in Figure 6.

Figure 6: Template for supervised fine-tuning.

Training Template

Question:{Question}
Answer: Let's think Step by Step.
{Rationale}
{Final Answer}

For all models, we employed full-parameter fine-tuning using the Adam optimizer. We set the warmup ratio to 0.1 and implemented a cosine learning rate scheduler. The number of epochs, learning rate, and batch size were model-specific; these hyperparameters are detailed in Table 6.

Table 6: The statistics of synthesized data.

Model	Batch size	Epoch	Learning Rate
Llama3-8B-Base	128	1	1e-5
Mistral-7B-Base	128	3	2e-6
DeepseekMath-7B-Base	128	3	2e-6
Llama3-70B-Base	128	1	1e-5

B.2 Evaluation

In our evaluation process, we employ greedy decoding and apply the same template used during training. We assess performance using exact match criteria, wherein results are considered correct only if they precisely match the correct answer or satisfy predefined rules for equivalence.

B.3 Baseline

In §4.2, our analysis centers on various synthesized datasets and fine-tuned models for generating mathematical instruction-tuning data. We compare the following approaches: WizardMath, MetaMath, MMIQC, MathScale, and Xwin-Math.

- WizardMath enhances existing data using the Evol-Instruct method, which modifies question difficulty. It also employs reinforcement learning to further improve model performance.
- MetaMath introduces techniques for rephrasing questions and implementing backward reasoning to expand existing datasets.
- MMIQC iteratively increases the complexity of existing questions to generate more challenging problems.
- MathScale extracts knowledge points from existing data and uses these as seed information for large-scale data synthesis.
- Xwin-Math prompts the language model to generate entirely new math problems based on existing ones.

We conducted our own training and testing for MetaMath and MMIQC, as their datasets are open-source. For WizardMath, MAmmoTH2, MathScale, and Xwin-Math, we evaluated their performance using the open-sourced models.

In §5.1, we evaluate several synthesis methods, including NumReplace, XwinMath, MMIQC, and RFT. XwinMath and MMIQC were introduced previously.

- NumReplace involves replacing numbers in existing problems without verification to generate new problems.
- RFT stands for rejection fine-tuning, where sampling rationales for existing problems, filter incorrect rationales with incorrect answers, followed by fine-tuning the model.

B.4 Benchmark

We selected five challenging benchmarks to evaluate mathematical problem-solving capabilities: the MATH test set, GaoKao-MATH, OlympiadBench-MATH, MathOdyssey, and TheoremQA.

- The MATH test set mirrors the distribution of the MATH training set, offering a comprehensive range of high-difficulty problems.
- GaoKao-MATH comprises 508 mathematics problems from China's college entrance examination (Gaokao).
- MathOdyssey features 387 advanced mathematics problems from university and high school levels, serving as the problem set for the 2024 Global AI Competition (GAIC) mathematics contest.
- OlympiadBench-MATH consists of 675 Olympiad-level mathematics competition problems, limited to text-based questions.
- TheoremQA includes 800 problems spanning mathematics, physics, and economics, requiring the application of domain-specific theorems for solution.

In Appendix F, we evaluate our methods on additional mathematical benchmarks, including AGIEval-Math (Zhong et al., 2023), AQuA (Ling et al., 2017), ASDiv (Miao et al., 2020), DeepMind Mathematics (Saxton et al., 2019), GSM8k (Cobbe et al., 2021), MMLU-Math (Hendrycks et al., 2020), and SVAMP (Patel et al., 2021).

- AGIEval-Math, a subset of AGIEval, contains 220 examples assessing mathematical abilities.
- AQuA (Algebra Question Answering) comprises 254 algebraic word problems.
- ASDiv (Academia Sinica Diverse MWP Dataset) contains 122 simple mathematics problems.
- DeepMind Mathematics is a diverse collection of problems designed to test fundamental mathematical capabilities across various domains.
- The GSM8k test set consists of 1,319 elementary-level mathematics word problems, evaluating multi-step problem-solving and basic arithmetic operations.
- MMLU-Math, part of the MMLU benchmark, assesses proficiency in various mathematical topics, including algebra, geometry, and statistics.
- SVAMP (Simple Variations on Arithmetic Math Word Problems) tests model robustness and adaptability by introducing slight variations to existing mathematics word problems.

C Synthesized Examples

We provide an example of a synthesized new problem, along with its original mathematical problem, natural language algorithm, and code function. Through this example, we can intuitively understand the intermediate results of our data synthesis and the final data.

Original Mathematical Problem

Question:
A printer prints 17 pages per minute. How many minutes will it take to print 200 pages? Express your answer to the nearest whole number.

Solution:
$\frac{200 \text{ pages}}{17 \text{ pages per minute}} \approx \boxed{12}$ minutes.

Final Answer:
12

Natural Language Algorithm

Algorithm Objective
Given:
- A printer prints p pages per minute.
- The desired number of pages to be printed is n.
Find:
- Approximate number of minutes needed to print $\boxed{n}$ pages.

Algorithm Process
1. **Calculate the Number of Minutes Needed:**

$$\text{Number of minutes} = \frac{\text{Total number of pages}}{\text{Pages per minute}}$$

$$\text{Number of minutes} = \frac{n}{p}$$

Placeholder Variables:
{"p": 17, "n": 200}

Constraints:
None 12

Code Function

```
def print_time(p, n):
    minutes = round(n / p)
    return minutes
```

Synthesized New Problem

Question:
A printer produces 24 pages per minute. How many minutes will it take to print 240 pages? Round your answer to the nearest whole number.

Solution:
240/24=10. So it will take 10 minutes to print 240 pages.

Final Answer:
10

D Prompts

D.1 Prompts of AIC

The AIC prompt is mainly divided into four parts. The first is used to extract the algorithm from the problem, the second is to implement the algorithm as code, the third generates possible contexts for the algorithm, and the fourth uses the context to contextualize the algorithm into problems and solutions.

Abstraction(Part 1)

Role
You are a professional mathematics teacher.
Your task is to design a question and the corresponding solution template and implement this solution template through a Python function.

Instructions
We will provide a specific mathematical problem along with its corresponding solution.
Based on this provided problem and answer, you are required to design a generic template for both the problem and its solution. A math problem usually contains some specific integers, such as constants in an expression or coefficients in a term of an equation, and changing these integers does not affect the approach to solving the problem. These specific numbers in the problem can be converted into some placeholder variables to form a problem template. You **must not** replace the condition except the specific integers in the question.

Task
The template needs to include four parts:
- **Question template**: Design a question template based on the problem and its solution. This template should replace **specific integers** in the question with **placeholder variables**.
- First, you need to describe the design method for the question template, including which specific integers in the question have been replaced with which placeholders.
- Then give me the list of the placeholder variables.
- Next, write out your template question.
- Furthermore, considering that the specific integers in the question may need to satisfy certain relationships, you should provide the constraints on the values of these placeholders.
- **Solution template**: Design a solution template based on the problem and its solution.

Abstraction(Part 2)

Examples
Question
The equation of the circle that passes through $(-1, 6)$ and which has a center at $(2, 3)$ can be written as $x^2 + y^2 + Ax + By + C = 0$. Find $A \times B \times C$.
Solution
As the center of the circle is at the point $(2, 3)$, and one point on the circle is at the point $(-1, 6)$, by the distance formula, the radius of the circle is $\sqrt{(2-(-1))^2 + (3-6)^2} = \sqrt{3^2 + 3^2} = \sqrt{18}$. The equation of the circle is then given by $(x-2)^2 + (y-3)^2 = 18$, and expanding,

$$x^2 - 4x + 4 + y^2 - 6y + 9 - 18 = 0$$

$$\implies x^2 + y^2 - 4x - 6y - 5 = 0.$$

Thus, $A \times B \times C = -4 \times -6 \times -5 = \boxed{-120}$.

Template
Question Template
Description
We replace -1, 6, 2, 3 with x_1, y_1, h, k to get the template of this question.
Placeholder Variables
```json
{"x_1": -1, "y_1": 6, "h": 2, "k": 3}
```
Template Question
Given:
- A circle passes through a point (x_1, y_1) and has its center at (h, k).
- The general equation of a circle can be written as $x^2 + y^2 + Ax + By + C = 0$.
Find:
- The product $A \times B \times C$ based on the given points.
Constraints
- Since the point through which the circle passes and its center cannot be the same point, (x_1, y_1) is not equal to (h, k).
Solution Template
1. **Calculate the Radius:** Use the distance formula with the given points (x_1, y_1) and (h, k) to find the radius r:

$$r = \sqrt{(h - x_1)^2 + (k - y_1)^2}$$

2. **Form the Equation of the Circle:** Based on the center (h, k) and radius r, the equation is:

$$(x - h)^2 + (y - k)^2 = r^2$$

3. **Expand and Rearrange:** Expand the equation and rearrange it into the general form to find values of A, B, and C:

$$x^2 + y^2 + Ax + By + C = 0$$

4. **Calculate $A \times B \times C$:** Substitute the found values of A, B, and C into the formula to find the product.

Implementation

Role
You are a professional mathematics teacher.

Instructions
You need to design **only one** Python function to implement the solution template. The parameters of this Python function are the placeholder variables in the question template, and the return value is the answer to the question.

Examples
Template
Question Template
Description
We replace -1, 6, 2, 3 with x_1, y_1, h, k to get the template of this question.
Placeholder Variables
“‘json
{"x_1": -1, "y_1": 6, "h": 2, "k": 3}
“‘
Template Question
Given:
- A circle passes through a point (x_1, y_1) and has its center at (h, k).
- The general equation of a circle can be written as $x^2 + y^2 + Ax + By + C = 0$.
Find:
- The product $A \times B \times C$ based on the given points.
Constraints
- Since the point through which the circle passes and its center cannot be the same point, (x_1, y_1) is not equal to (h, k).
Solution Template
1. **Calculate the Radius:** Use the distance formula with the given points (x_1, y_1) and (h, k) to find the radius r:

$$r = \sqrt{(h - x_1)^2 + (k - y_1)^2}$$

2. **Form the Equation of the Circle:** Based on the center (h, k) and radius r, the equation is:

$$(x - h)^2 + (y - k)^2 = r^2$$

3. **Expand and Rearrange:** Expand the equation and rearrange it into the general form to find values of A, B, and C:

$$x^2 + y^2 + Ax + By + C = 0$$

4. **Calculate $A \times B \times C$:** Substitute the found values of A, B, and C into the formula to find the product.

Python Code
“‘Python
import math
def circle_equation(x1, y1, h, k):
    if x1==h and y1==k:
        raise ValueError("(x_1, y_1) shouldn’t equal to (h, k)")
    r = math.sqrt((h - x1)**2 + (k - y1)**2)
    A = -2 * h
    B = -2 * k
    C = h**2 + k**2 - r**2
    result = A * B * C
    return result
“‘

The contextualization of a natural language algorithm into a concrete problem involves two primary steps. Initially, the synthesizer generates diverse placeholder values based on the problem through multiple sampling and deduplication processes.

Contextualization 1

Role
- You are an experienced mathematics teacher.
- You need to generate mathematical problems for students based on a mathematical template.

Instructions
- You will be provided with a problem template, which consists of the following parts:
- Template Question: Represents a template for a class of mathematical problems, containing placeholder variables.
- Placeholder variables: When placeholder variables are given specific values, the template question becomes a specific problem with a determinable final answer.
- Constraints: Only when the placeholder variables satisfy these constraints can they become a mathematically logically consistent problem.
- You need to sample specific values for the placeholder variables based on the constraints on them.

Examples
Template Question
Given:
- A circle passes through a point (x_1, y_1) and has its center at (h, k).
- The general equation of a circle can be written as $x^2 + y^2 + Ax + By + C = 0$.
Find:
- The product $A \times B \times C$ based on the given points.
Placeholder variable

$$"x_1", "y_1", "h", "k"$$

Constraints
- Since the point through which the circle passes and its center cannot be the same point, (x_1, y_1) is not equal to (h, k).
Placeholder Values
“‘json {"x_1": -1, "y_1": 6, "h": 2, "k": 3} “‘

Template Question
Given:
- A circle passes through a point (x_1, y_1) and has its center at (h, k).
- The general equation of a circle can be written as $x^2 + y^2 + Ax + By + C = 0$.
Find:
- The product $A \times B \times C$ based on the given points.
Placeholder variable
["x_1", "y_1", "h", "k"]
Constraints
- Since the point through which the circle passes and its center cannot be the same point, (x_1, y_1) is not equal to (h, k).
Placeholder Values
“‘json
{"x_1": -1, "y_1": 6, "h": 2, "k": 3}
“‘

Subsequently, these placeholder values, serving as context, are utilized to prompt the synthesizer to transform the algorithm into a specific problem and its corresponding rationale.

Contextualization 2

Role
- You are an experienced mathematics teacher.

Instructions
- You will be provided with a problem template, which consists of the following parts:
- Template Represents a template for a class of mathematical problems, containing placeholder variables.
- Placeholder variables values: When placeholder variables are given specific values, the template question becomes a specific problem with a determinable final answer.
- You need to generate a discrete mathematical problem and solution accroding to the template question and solution and the values of its placeholder variables.

Examples ## Template Question
Given:
- $f(a) = b$
- $f^{-1}(x + c) = df^{-1}(x) + e$ for all x,
Find:
- $f^{-1}(y)$.
Solution Template
1. **Determine $f^{-1}(b)$**: Using the known value $f(a) = b$, derive $f^{-1}(b) = a$.
2. **Iterate Using the Functional Equation**: Use the equation $f^{-1}(x + c) = df^{-1}(x) + e$ iteratively starting from $x = b$ to calculate $f^{-1}(b + c), f^{-1}(b + 2c), \ldots, f^{-1}(y)$ by applying:

$$f^{-1}(x + c) = df^{-1}(x) + e$$

until $x + c$ reaches y.
3. **Result**: The value computed in the final iteration will be $f^{-1}(y)$.
Placeholder Values
“‘json
{"a": 2, "b": 5, "c": 4, "d": 2, "e": 1, "y": 17}
“‘
Question
The equation of the circle that passes through $(-1, 6)$ and which has a center at $(2, 3)$ can be written as $x^2 + y^2 + Ax + By + C = 0$. Find $A \times B \times C$.
Solution
As the center of the circle is at the point $(2, 3)$, and one point on the circle is at the point $(-1, 6)$, by the distance formula, the radius of the circle is $\sqrt{(2-(-1))^2 + (3-6)^2} = \sqrt{3^2 + 3^2} = \sqrt{18}$. The equation of the circle is then given by $(x - 2)^2 + (y - 3)^2 = 18$, and expanding,

$$x^2 - 4x + 4 + y^2 - 6y + 9 - 18 = 0$$
$$\Longrightarrow x^2 + y^2 - 4x - 6y - 5 = 0.$$

Thus, $A \times B \times C = -4 \times -6 \times -5 = \boxed{-120}$.

E Failure Case

Our analysis revealed the specific data points that failed to pass the verification criteria, encompassing both those that did not pass the code verification in stage 1 and those that failed the subsequent new problem verification in stage 2.

E.1 Failure Cases in Stage 1

As illustrated in the figure below, the synthesizer may err when abstracting the original question into an algorithmic objective. The equations $a^5 = b^4$ and $c^3 = d^2$ describe relationships between four variables. The synthesizer abstracts these as $A^5 = B^4$ and $C^3 = D^2$, treating A, B, C, and D as placeholder variables. However, synthesizing a new problem by assigning other expressions to A, B, C, and D is not feasible.

Failure Case 1

Question:
Assume that a, b, c, and d are positive integers such that $a^5 = b^4$, $c^3 = d^2$, and $c - a = 19$. Determine $d - b$.

Solution:
It follows from the givens that a is a perfect fourth power, b is a perfect fifth power, c is a perfect square and d is a perfect cube. Thus, there exist integers s and t such that $a = t^4$, $b = t^5$, $c = s^2$ and $d = s^3$. So $s^2 - t^4 = 19$. We can factor the left-hand side of this equation as a difference of two squares, $(s - t^2)(s + t^2) = 19$. 19 is a prime number and $s + t^2 > s - t^2$ so we must have $s + t^2 = 19$ and $s - t^2 = 1$. Then $s = 10, t = 3$ and so $d = s^3 = 1000$, $b = t^5 = 243$ and $d - b = \boxed{757}$.

Algorithm Objective:
Assume that A, B, C, and D are positive integers such that $A^5 = B^4$, $C^3 = D^2$, and $C - A = n$.

Algorithm Process:
It follows from the givens that A is a perfect fourth power, B is a perfect fifth power, C is a perfect square and D is a perfect cube. Thus, there exist integers s and t such that $A = t^4$, $B = t^5$, $C = s^2$ and $D = s^3$. So $s^2 - t^4 = n$. We can factor the left-hand side of this equation as a difference of two squares, $(s - t^2)(s + t^2) = n$. n is a prime number and $s + t^2 > s - t^2$ so we must have $s + t^2 = n$ and $s - t^2 = 1$. Then $s = n - 1$, $t = 3$ and so $D = s^3 = (n - 1)^3$, $B = t^5 = 243$ and $D - B = \frac{(n-1)^3 - 243}{1}$.

Python Code:
```
def find_solution(A, B, C, D):
    n = C - A
    t = 3
    s = n-1
    D = (n-1)**3
    B = t**5
    result = (n-1)**3 - t**5
    return result
```

Execution Result: -244

Answer from Original Problem: 755

Even when placeholder variables are correctly identified and parameterized, errors may still arise during the abstraction of the original problem's rationale into an algorithmic process. This can result in the generation of an incorrect algorithm, independent of any errors that may occur during the initial problem abstraction into an algorithmic objectiveness.

Failure Case 2

Question:
Jar A has exactly four red buttons and eight blue buttons. Carla then removes the same number of red buttons as blue buttons from Jar A and places them in an empty Jar B. Jar A now has $\frac{2}{3}$ of its original number of buttons. If Carla were now to randomly choose a button from Jar A and a button from Jar B, what is the probability that the two chosen buttons will both be red? Express your answer as a common fraction.

Solution:
Two-thirds of Jar A's original $4 + 8 = 12$ buttons is 8 buttons. Therefore, four buttons were removed from Jar A: two red buttons and two blue buttons. So the probability that the button drawn from Jar A is red is $\frac{2}{8}$ and the probability that the button drawn from Jar B is red is $\frac{2}{4}$. Therefore, the probability that both buttons are red is $\frac{2}{8} \cdot \frac{2}{4} = \boxed{\frac{1}{8}}$.

Algorithm Objective:
Jar A has exactly a red buttons and b blue buttons. Carla then removes the same number of red buttons as blue buttons from Jar A and places them in an empty Jar B. Jar A now has $\frac{2}{3}$ of its original number of buttons. If Carla were now to randomly choose a button from Jar A and a button from Jar B, what is the probability that the two chosen buttons will both be red? Express your answer as a common fraction.

Algorithm Process:
Two-thirds of Jar A's original $a + b$ buttons is $a + b/3$ buttons. Therefore, $(a + b)/3$ buttons were removed from Jar A: $a/2$ red buttons and $b/2$ blue buttons. So the probability that the button drawn from Jar A is red is $\frac{a/2}{a+b/3}$ and the probability that the button drawn from Jar B is red is $\frac{a/2}{a/2}$. Therefore, the probability that both buttons are red is $\frac{a/2}{a+b/3} \cdot \frac{a/2}{a/2} = \frac{1}{8}$.

Python Code:

```
def probability_of_both_red_buttons(a, b):
    removed_buttons = (a + b) / 3
    a_removed = a // 2
    b_removed = b // 2
    probability_from_A = a_removed / (a + b / 3)
    probability_from_B = a_removed / (a_removed)
    return probability_from_A * probability_from_B
```

Execution Result: 0.3

Answer from Original Problem: 1/8

In addition to the aforementioned two error types, the synthesizer may encounter implementation errors when translating the abstract algorithm into concrete code. In such cases, while the abstraction successfully captures the problem's algorithmic essence, the resulting implementation fails to accurately represent the intended algorithm.

E.2 Failure Cases in Stage 2

The causes of errors in the second stage are multifaceted. Case 3 exemplifies one such scenario: the synthesizer transforms the abstract algorithm into a concrete problem, but the resulting mathematical expression becomes ambiguous, and the problem's logic lacks coherence. This issue may arise from two potential sources: either the algorithm itself is flawed but somehow passed the initial checks, or the contextualization process incorrectly transformed a sound algorithm into a faulty problem.

Failure Case 3

Question:
What is the positive difference between the sum of all the positive integers from 1 to 70 and the sum of the 100 positive integers from 1 to 70, if every integer in the latter sum is rounded to its nearest multiple of 10 (rounding 5s up)?

Solution:
From 1 to 10, Kate's sum is $10 + 20 + 30 + 40 + 50 + 60 + 70 = 350$. Jo's sum is $1+2+3+4+5+6+7+8+9+10 = 55$. The difference is $350 - 55 = 295$. As the same logic applies to every ten numbers greater than 20, multiply the difference per 10 numbers by $\frac{70}{10} = 7$ to find the total difference: $7 \times 295 = 2065$. The sum of the first 70 positive integers is $\frac{70 \times 71}{2} = 2485$. Subtract the difference to find Kate's sum: $2485 - 2065 = 420$. The positive difference is thus $2485 - 420 = \boxed{2065}$.

Code Answer: 35

This example illustrates another instance where the problem failed the verification process. A thorough examination reveals that while the problem's logic is sound, the solution procedure is correct, and the final answer is accurate, the code output yielded an incorrect result. This discrepancy may be attributed to an error in the code function.

Failure Case 4

Question:
If $\frac{q}{r} = 2$, $\frac{s}{r} = 3$, and $\frac{s}{t} = 4$, then what is $\frac{t}{q}$?

Solution:
From the given information,

$$\frac{s}{r} \cdot \frac{r}{q} \cdot \frac{t}{s} = \frac{3}{1} \cdot \frac{1}{2} \cdot \frac{1}{4}$$

$$\frac{t}{q} = \frac{3}{8} = \boxed{\frac{3}{8}}$$

Code Answer: 9.0

F Result on More Benchmarks

Table 7: Result on more mathematical benchmarks.

Models	AGIEval-Math	AQuA	ASDiv	DeepMind	GSM8k	MMLU-Math	SVAMP
Llama3-70B-Base							
Llama3-70B-Seed	26.5	24.1	93.4	44.2	76.1	35.7	83.3
Llama3-70B-MetaMath	51.4	26.1	**98.4**	49.1	**85.7**	36.7	83.0
Llama3-70B-Abel	47.3	26.5	95.9	45.7	84.5	38.6	87.3
Llama3-70B-AIC	**57.3**	**30.4**	95.9	**59.6**	81.3	**43.8**	**87.7**
DeepSeekMath-7B-Base							
DSMath-7B-Seed	43.6	26.1	89.3	42.0	56.2	34.3	66.3
DSMath-7B-MetaMath	49.1	26.5	96.5	48.4	78.7	36.0	79.3
DSMath-7B-Abel	46.8	25.3	95.9	41.4	72.4	35.5	81.0
DSMath-7B-AIC	**56.8**	**32.0**	**96.7**	**59.8**	**82.2**	**44.1**	**85.3**
Mistral-7B-Base							
Mistral-7B-Seed	29.5	13.4	87.7	15.9	39.0	22.6	63.7
Mistral-7B-MetaMath	40.5	18.6	94.3	24.6	**78.6**	32.2	**79.3**
Mistral-7B-Abel	35.0	20.6	94.3	18.3	73.4	27.0	77.7
Mistral-7B-AIC	**46.4**	**23.7**	**94.3**	**45.5**	69.2	**38.2**	73.7
Llama3-8B-Base							
Llama3-8B-Seed	29.5	15.4	84.4	24.7	42.7	24.2	67.7
Llama3-8B-MetaMath	39.5	19.8	96.7	36.1	**79.3**	30.8	**78.0**
Llama3-8B-Abel	34.5	16.2	93.4	25.2	75.8	31.7	75.7
Llama3-8B-AIC	**47.3**	**23.3**	**95.9**	**49.1**	66.8	**38.2**	74.0